

CE143: COMPUTER CONCEPTS & PROGRAMMING

July – November 2019

Chapter – 6

Looping

Objectives

- To learn about repetition (looping) control structures, i.e. while, for and do...while
- To Examine break and continue statements
- To discover how to form and use *nested* control structures

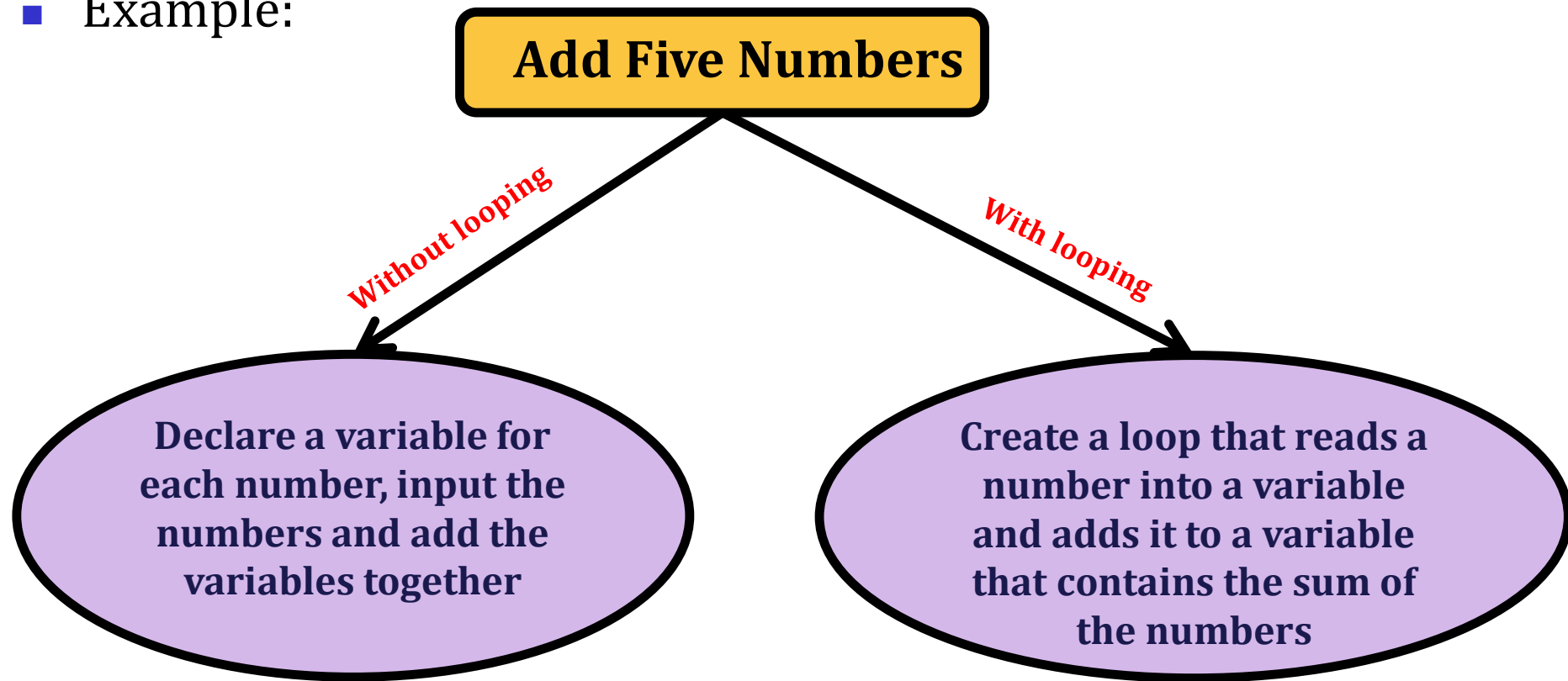
Introduction

In this chapter, we will discuss

- Need of looping
- (pre-test) entry-controlled loop: while, for, (post-test) exit-controlled loop: do...while, difference
- Use of sentinel values.
- Nesting of looping statements
- use of break & continue
- use of if...else in loop
- infinite loop.

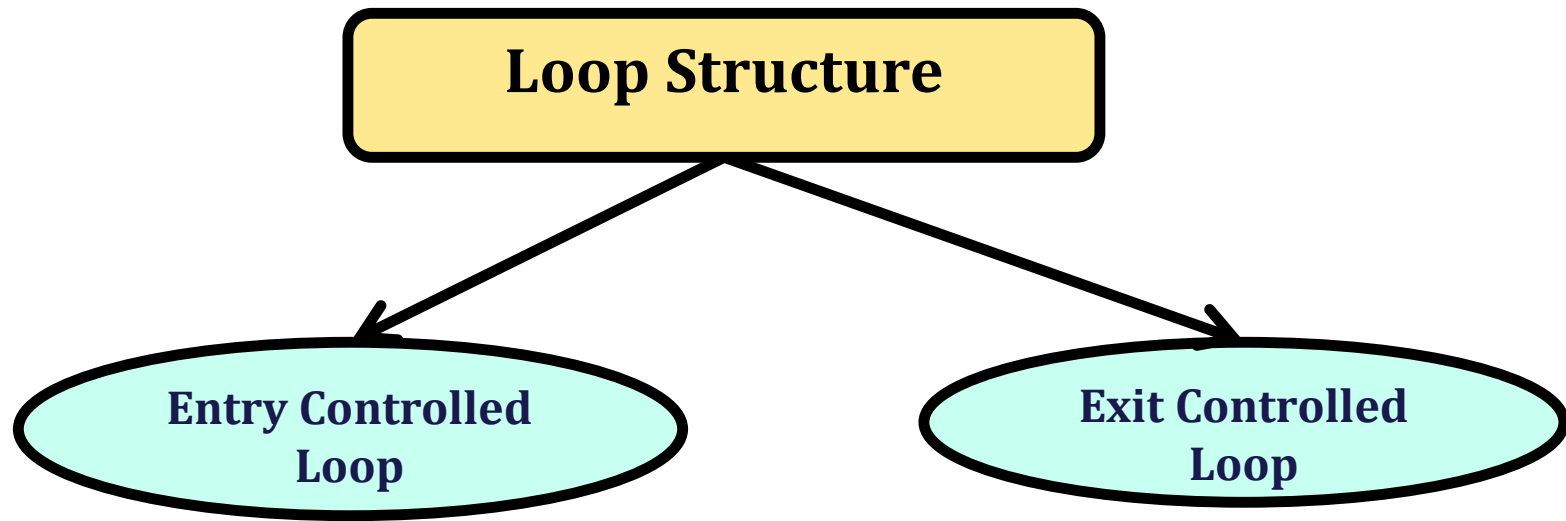
Need of Looping

- Allow you to efficiently use variables
- It can input, add, and average multiple numbers using a limited number of variables
- Example:



Loop Control Structures

- Depending on the position of the control statement in the loop, a control structure may be classified into two categories.



Loop Control Structures

Entry Controlled Loop

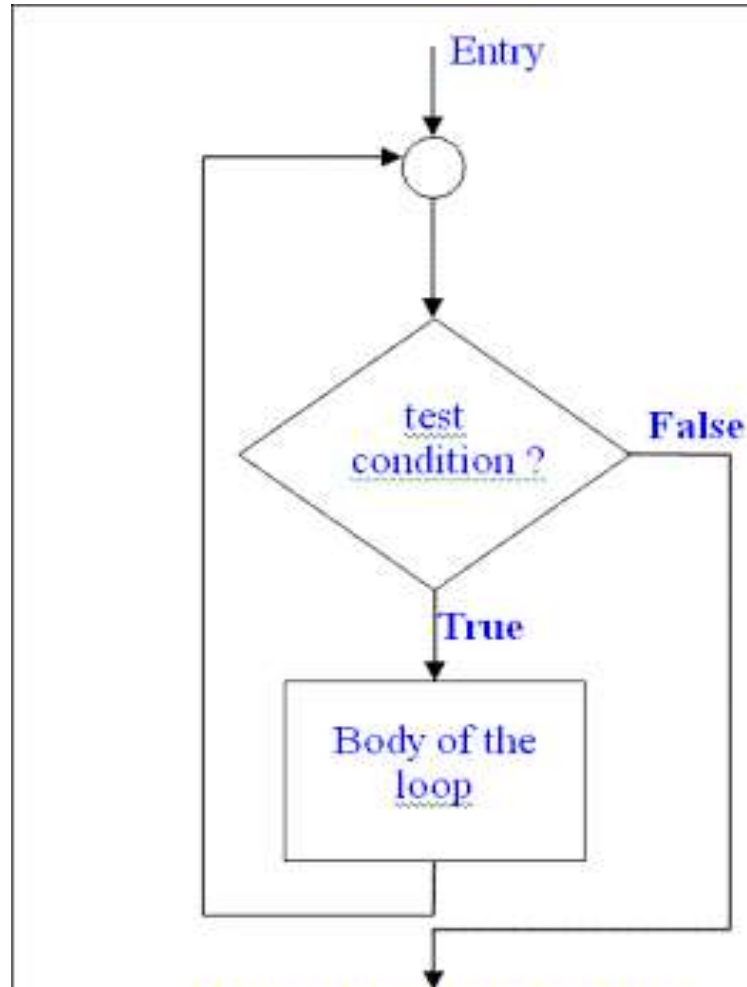


Figure : Entry controlled loop

Exit Controlled Loop

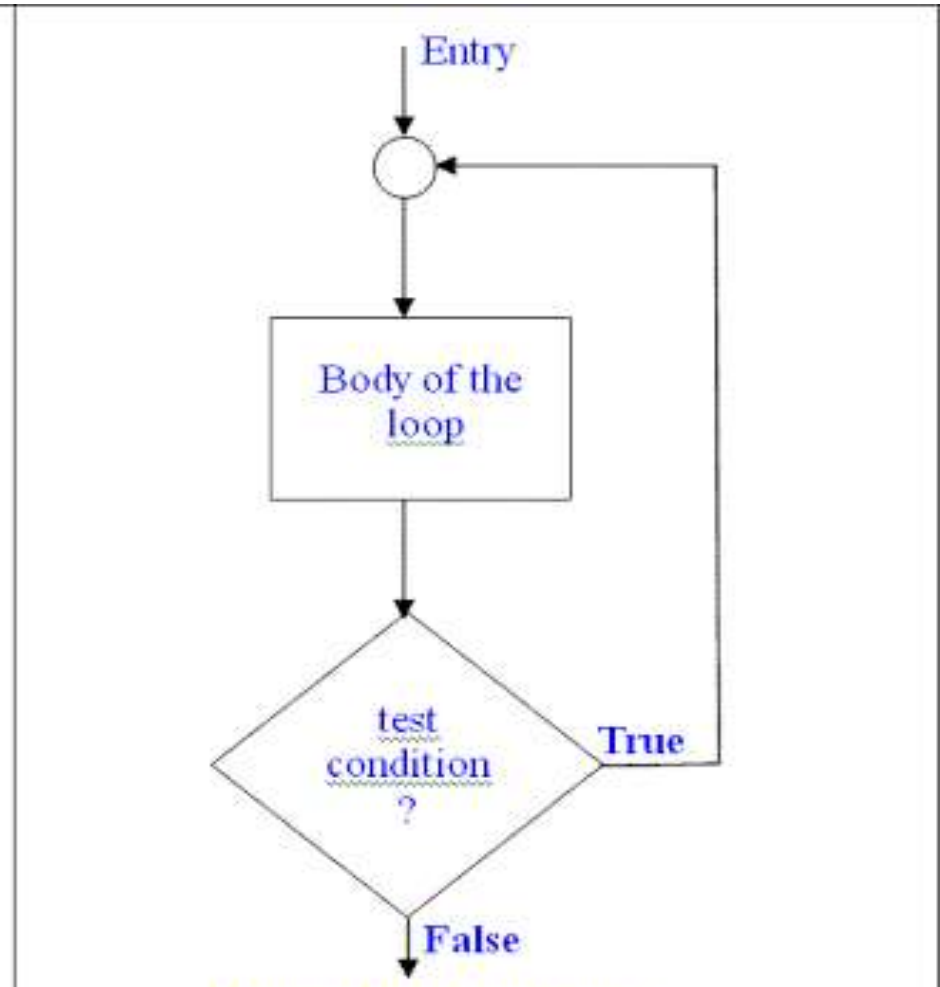
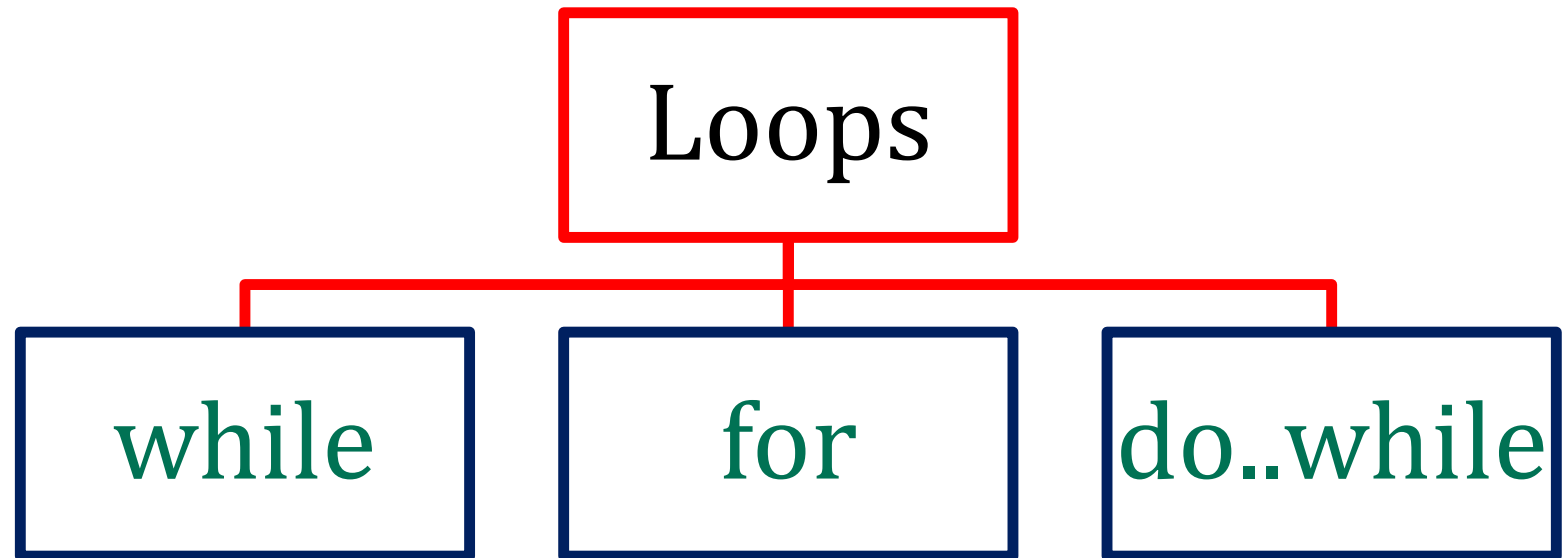


Figure : Exit controlled loop

Loop Control Structures

- A looping process, would include the following steps.
 1. **Setting and initialization of a condition variable**
 2. **Execution of the statements in the loop**
 3. **Test for a specified value of the condition variable for execution of the loop.**
 4. **Incrementing or updating the condition variable**

Loop Control Structures



Entry Controlled Loop-while loop

- The while loop is the simplest looping structure
- If we have to execute some statements repeatedly as long as certain condition become true we can use the while loop.
- In the while loop ,the given expression will be check at the time of the loop entry, if given expression becomes true then control will transfer into the loop body.
- The statement inside the loop body will be executed.
- The counter value will be modified and again given expression will be checked to enter in to the loop.
- This process will continue until the given expression become false.
- The while loop contains only expression part ,initialization will be done before the while loop

Entry Controlled Loop-while loop

Syntax

```
while(condition)
{

    //loop body
}
```

Example

```
while( i <= 10)
{

    printf("\n Value of I is %d", i) ;
    i++ ; // modify counter value
}
```

Entry Controlled Loop-while loop

```
/* program to print multiplication table using while loop. */
```

```
void main()
```

```
{
```

```
    int n, i, ans;
```

```
    clrscr();
```

```
    printf("\n Enter No. to print Table : ");
```

```
    scanf("%d", &n);
```

```
    i = 1 ;    // set counter value to 1
```

```
    while( i <= 10)
```

```
    {
```

```
        ans = n * i;
```

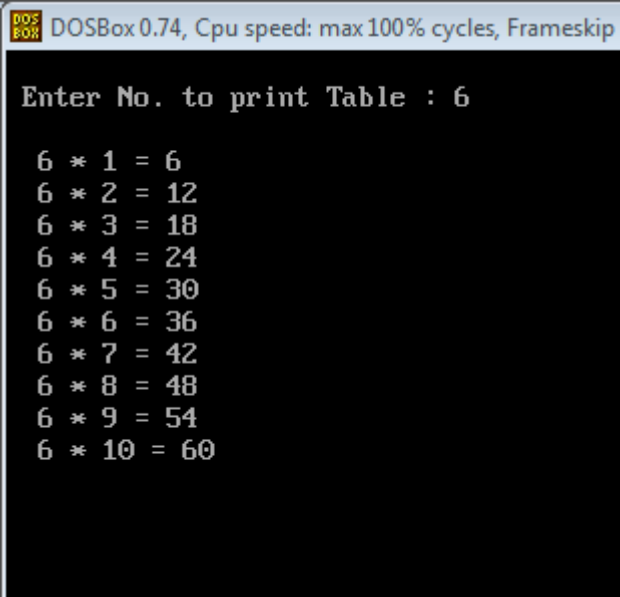
```
        printf("\n %d * %d = %d", n,i, ans) ;
```

```
        i++ ;    // modify counter value
```

```
    }
```

```
    getch();
```

```
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip

Enter No. to print Table : 6

6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
```

Entry Controlled Loop-while loop

The While Statement :

Example :

```
int a =1 ;
```

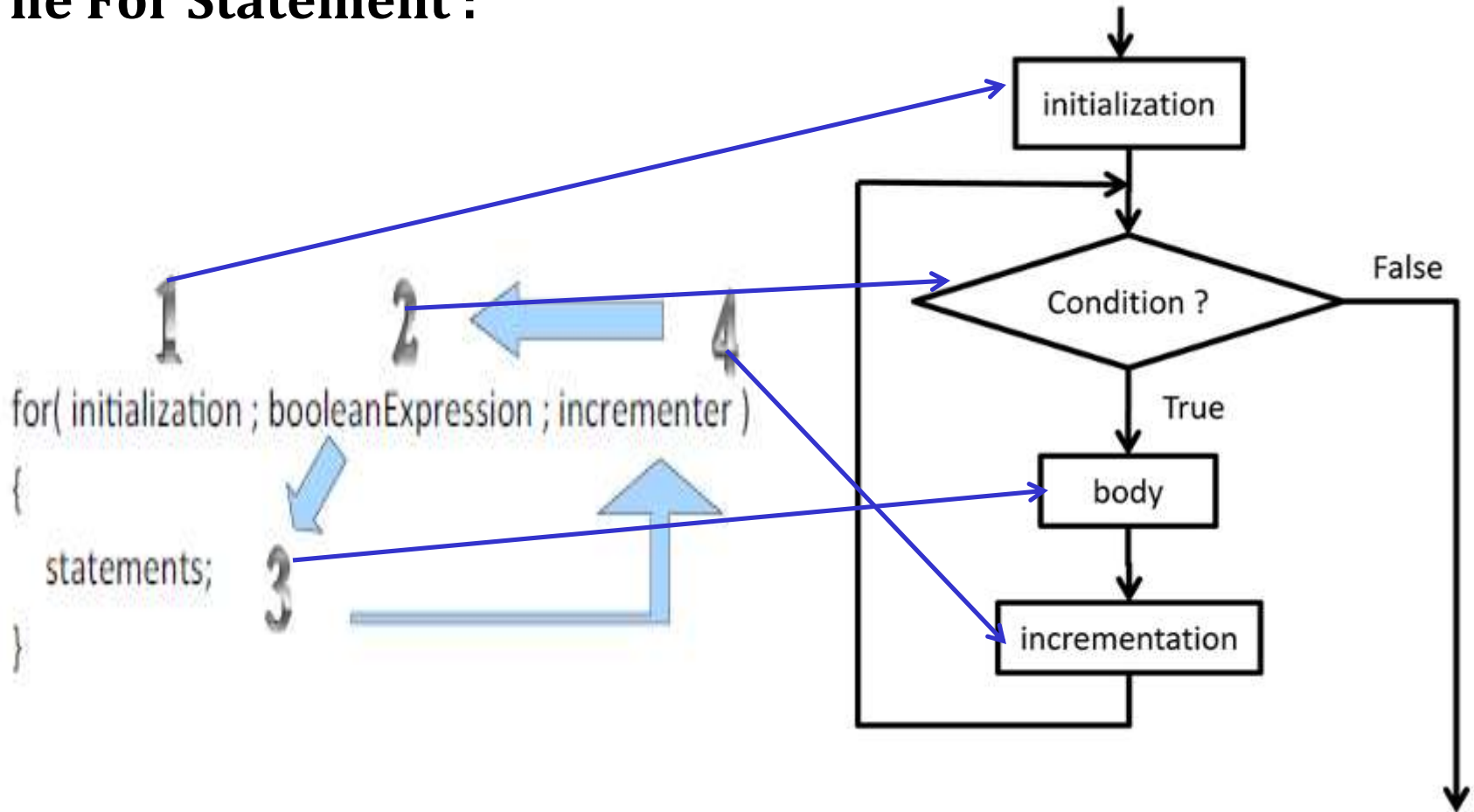
```
While (a <=5) // 1<=5 stands true
```

```
{  
    printf(“%d ”,a); // print 1  
    a++; // a=2  
}
```

Output : 1 2 3 4 5

Entry Controlled Loop-**for loop**

The For Statement :



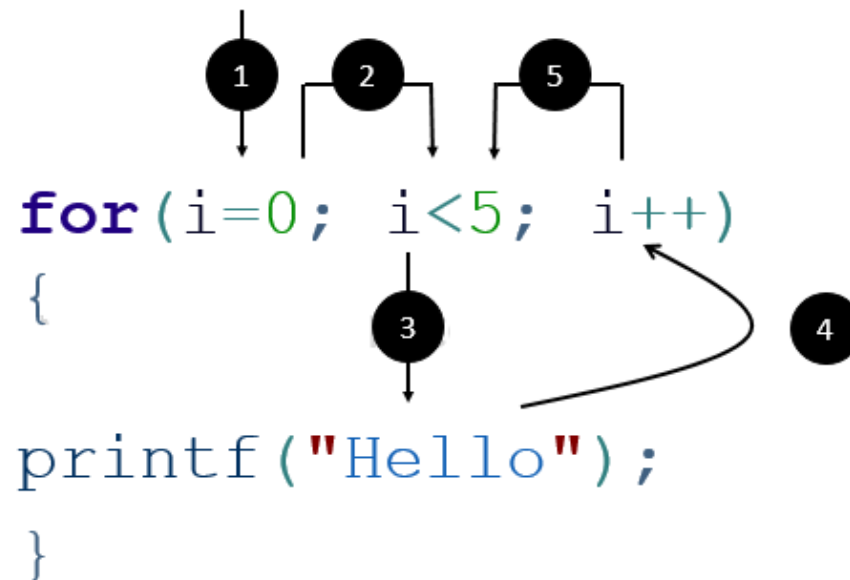
Entry Controlled Loop-**for loop**

- for-loops are *counter-controlled*, meaning that they are normally used whenever the number of iterations is known in advance.
- where again the body can be either a single statement or a block of statements within { curly braces }.
- For loop is the most powerful and flexible looping structure.
- We can perform any complex task using for loop.

Entry Controlled Loop-**for loop**

Syntax

```
for ( initialization ; condition ; inc/dec )  
{  
    // loop body  
}
```



Entry Controlled Loop-**for loop**

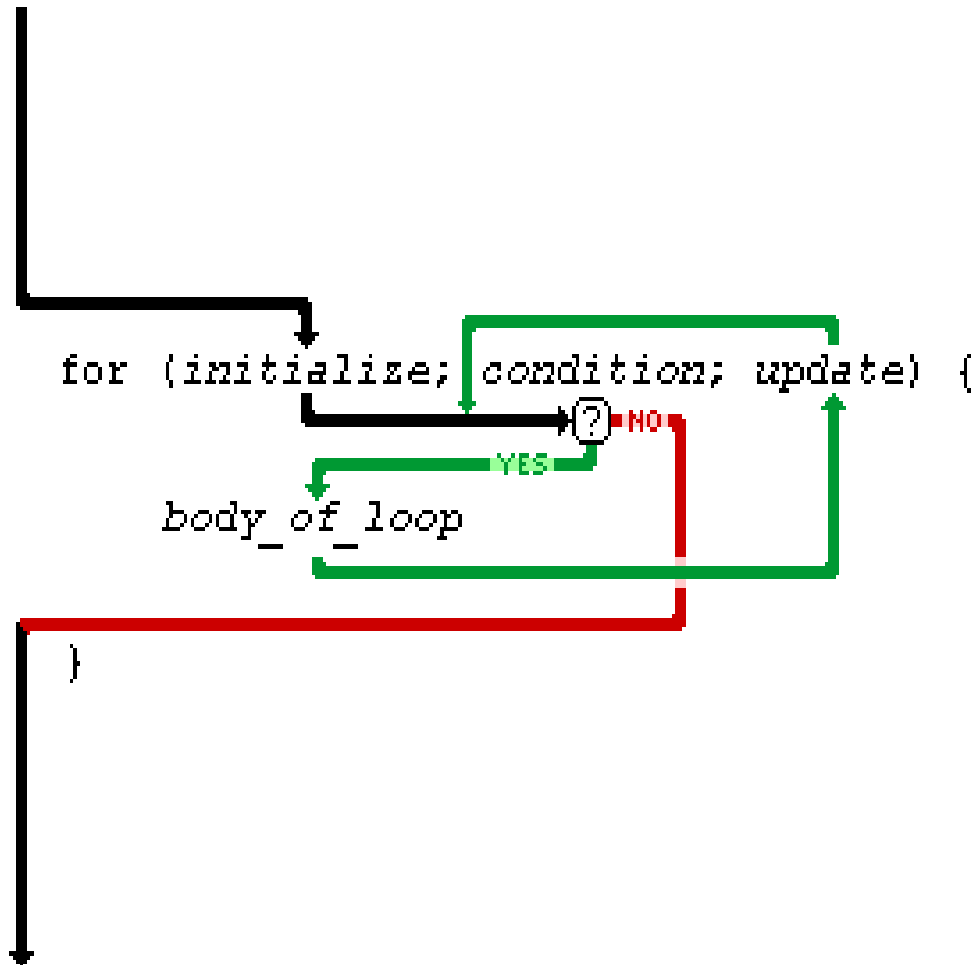
- For loop contains three different parts: **initialization, condition and inc/dec part.**
- When the loop executes for the first time, initialization of the counter variable will be done after that expression given into the condition part is tested, if test expression evaluate to true, control will enter into the loop body.
- The statements inside the loop will be executed.
- After that control will transfer to the third part where counter value will be either incremented or decremented.
- With the modified counter value test expression will be check again to enter into the loop.

Entry Controlled Loop-**for loop**

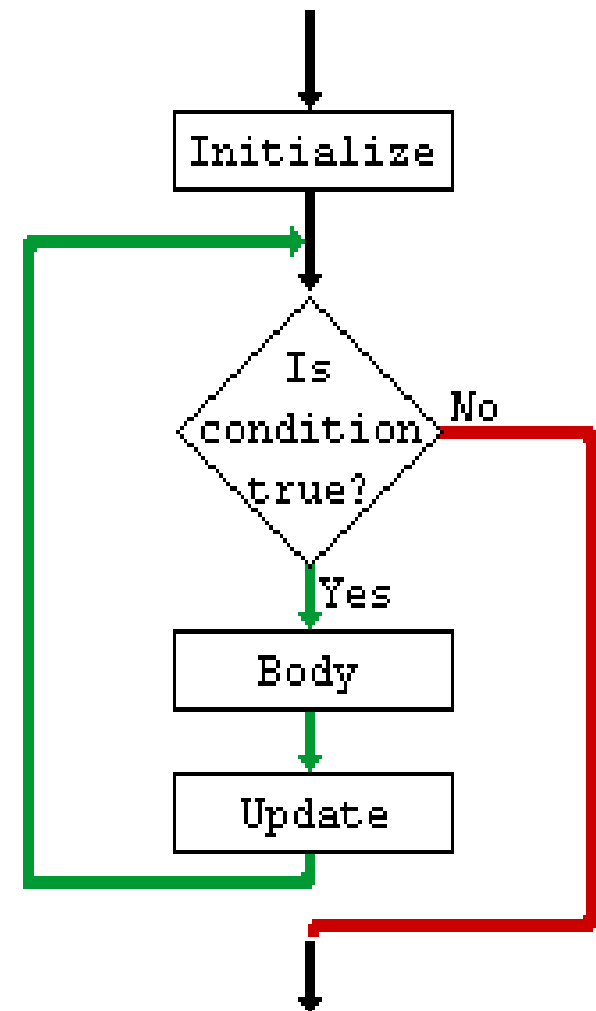
- This process will be repeated until condition becomes false.
- Once the condition becomes false, the loop will be skipped and statement following by for loop will be executed.
- The initialization will be performed only once when loop encounters first time.
- For loop also allows specify multiple condition as well as complex expression to be tested.
- We can also skip any of the part of the loop as per our requirement.
- for loop can also be executed either in straight forward or in reverse manner similar to while loop.

Entry Controlled Loop-**for loop**

Program Syntax

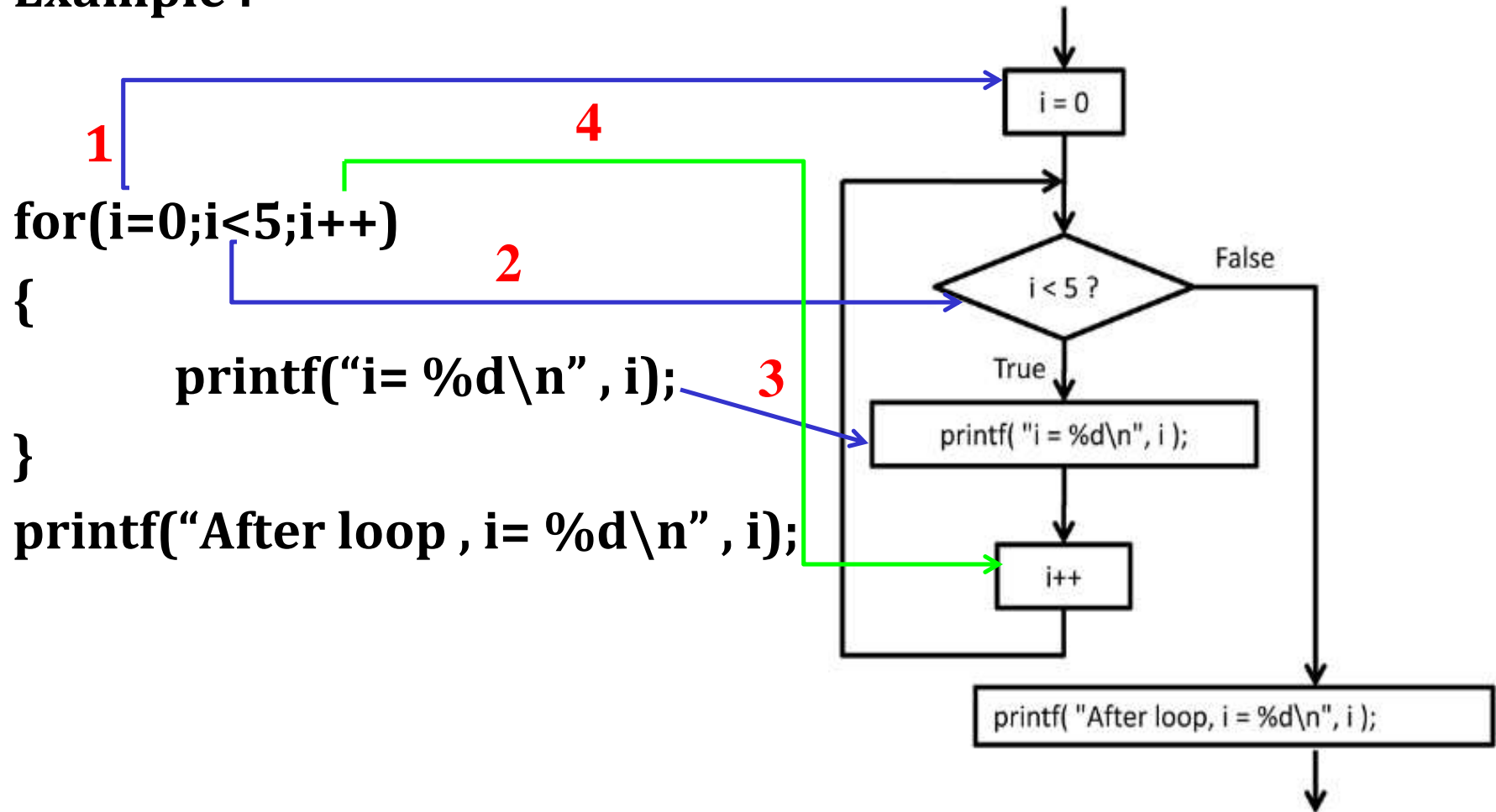


Structure Chart



Entry Controlled Loop-**for loop**

Example :

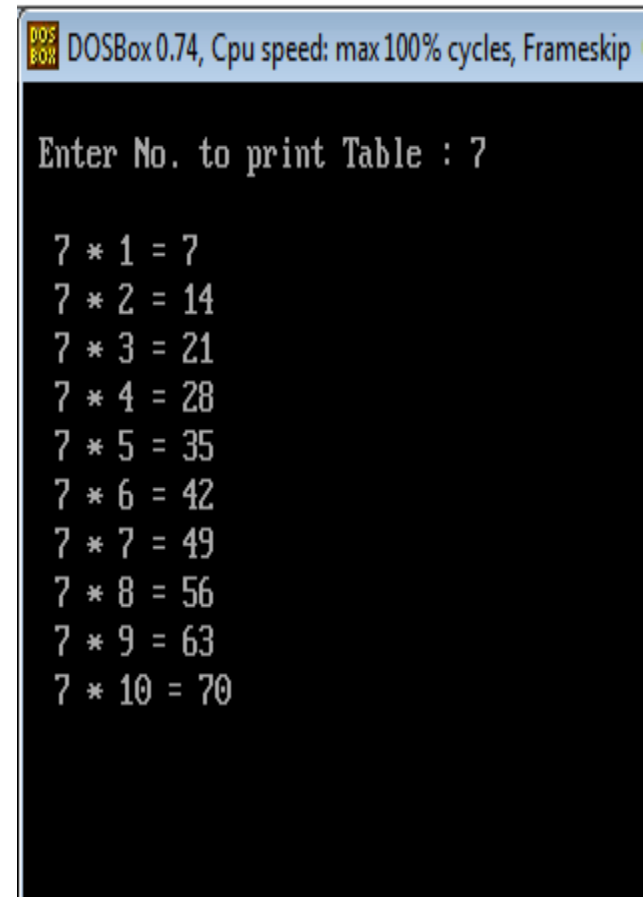


Entry Controlled Loop-**for loop**

```
void main()
{
    int n, i, ans;
    clrscr();

    printf("\n Enter No. to print Table : ");
    scanf("%d", &n);

    for( i = 1; i <=10 ; i++ )
    {
        ans = n * i;
        printf("\n %d * %d = %d", n,i, ans) ;
    }
    getch();
}
```



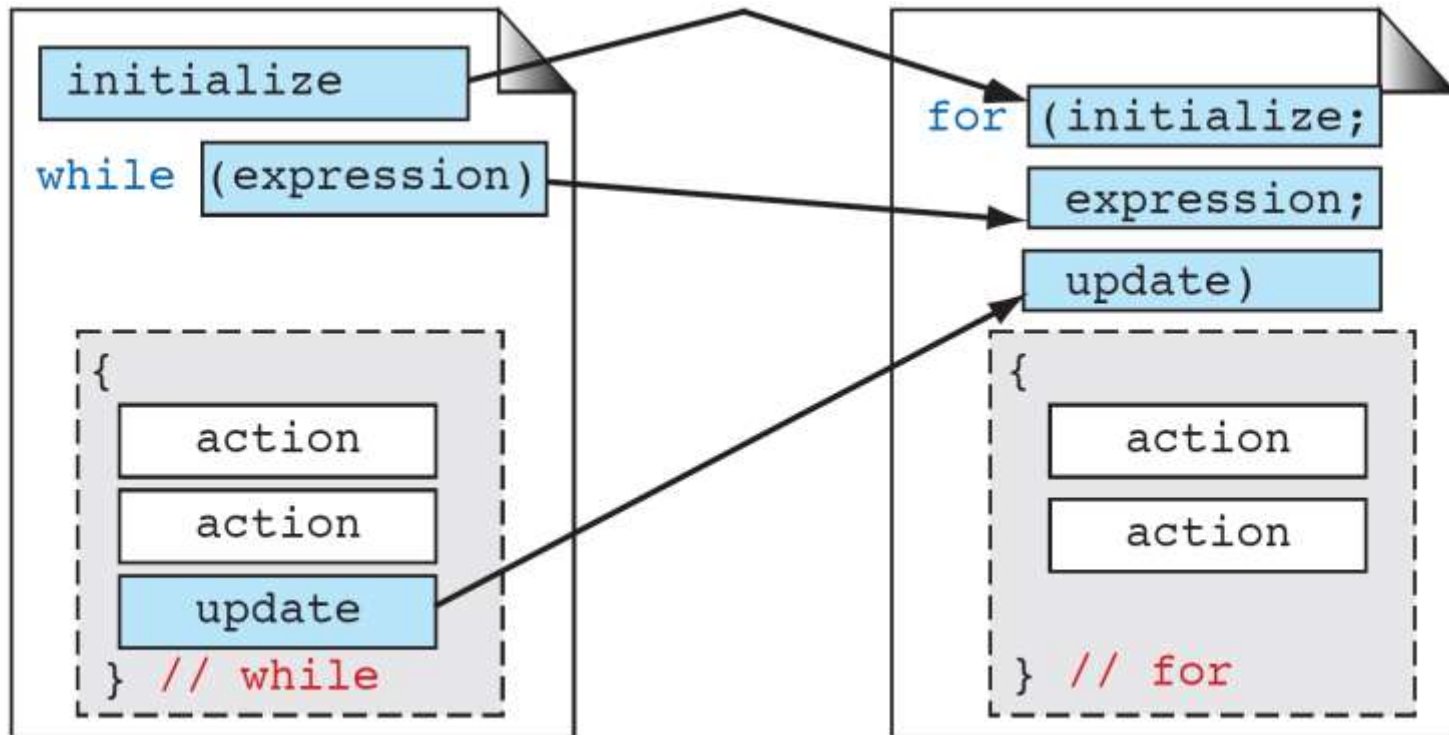
```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip

Enter No. to print Table : 7

7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

Entry Controlled Loop-**for loop**

Comparing *for* and *while* Loop



Entry Controlled Loop-**for loop**

Note

A for loop is used when a loop is to be executed a known number of times. We can do the same thing with a while loop, but the for loop is easier to read and more natural for counting loops.

Additional features of for loop

1. p=1;

for(n=0;n<17;++n)

Multiple initialization

2. for(p=1,n=0;n<17;++n)

Multiple initialization and
inc/dec

3. for(n=1,m=50;n<=m; n=n+1,m=m-1)

4. for i=1; i<20 && sum<100; ++i)

Complex conditions

5. for(x=(m+n)/2; x>0; x=x/2)

Multiple initialization
instead of inc/dec

Additional features of for loop

6. $m=5$;

```
for(;m!=100;){
printf(“%d\n”,m);
m=m+5;
}
```

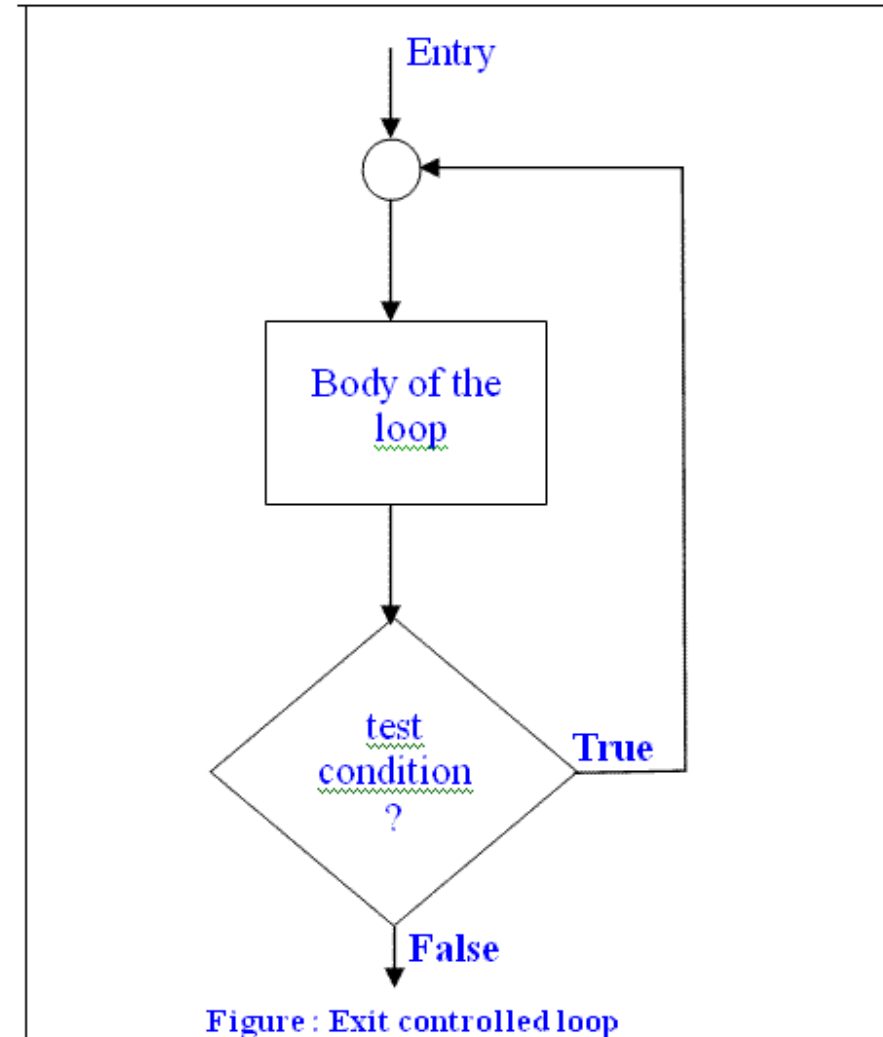
Initialization and inc/dec section can be omitted. (Initialization can be done before the for statement and inc/dec can be done inside the body loop)

```
7. for(j=1000;j>0;j=j-1)
    ;
```

- **Loop is executed 1000 times without producing any output; it simply causes time delay.**
- **Body of the loop contains only a semicolon, known as null statement**

Exit Controlled Loop-**do..while loop**

- An **Exit Control Loop** checks the condition for exit and if given condition for exit evaluate to true, control will exit from the loop body else control will enter again into the loop.
- Such type of loop controls exit of the loop that's why it is called **exit control loop**.



Exit Controlled Loop-**do...while loop**

- Do ...while loop is an exit control loop.
- It is a variation of while loop.
- When we want to execute or perform some task at least for once, we can use do...while loop structure.
- In a while loop if the expression becomes false at very first try then loop will never be executed.
- Since, do...while is of type exit control loop it checks the condition at last so, first time the loop will be execute unconditionally.
- If the condition for the exit becomes true then loop will be terminate otherwise it will be executed for the next time.

Exit Controlled Loop-**do...while** loop

Syntax

```
do  
{  
  //loop body  
} while (expression);
```

Example

```
do  
{  
    printf("\n Value of I is %d", i) ;  
    i++ ; // modify counter value  
} while( i <= 10);
```

Exit Controlled Loop-**do...while** loop

Comparision of While and Do..While Loop :

```
int main()
```

```
{
```

```
    bool a=true;
```



A diagram showing a code block for a while loop. A yellow speech bubble points to the condition 'false' in the while statement, containing the text 'Pretest nothing prints'.

```
while (false)
{
    printf("Hello World");
} // while
```



A diagram showing a code block for a do-while loop. A yellow speech bubble points to the condition 'false' in the while statement, containing the text 'Post-test Hello... prints'.

```
do
{
    printf("Hello World");
} while (false);
```

```
}
```

Exit Controlled Loop-**do..while** loop

```
void main()
{
    float x, y, ans;
    int choice;

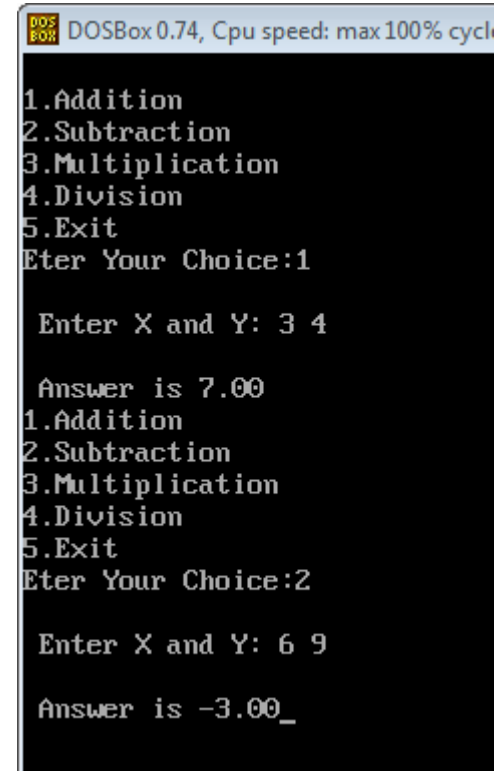
    do
    {

printf("\n1.Addition\n2.Subtractio
n\n3.Multiplication\n4.Division\n
5.Exit);

        printf("\nEnter Your Choice:");
        scanf("%d", &choice);

if(choice!=5)
{
    printf("\nEnter X and Y: ");
    scanf("%f %f", &x, &y);
}
```

```
switch(choice)
{
    case 1:
        ans = x + y;
        break;
    case 2:
        ans = x - y;
        break;
    case 3:
        ans = x * y;
        break;
    case 4:
        ans = x / y;
        break;
    case 5:
        exit(0);
    default:
        printf("\n Invalid choice!");
}
printf("\n Answer is %.2f", ans );
getch();
} while(choice!=5)
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycl

1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Exit
Enter Your Choice:1

Enter X and Y: 3 4

Answer is 7.00
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Exit
Enter Your Choice:2

Enter X and Y: 6 9

Answer is -3.00_
```

Exit Controlled Loop-**do...while loop**

The Do... While Loop :

Example :

```
int main()
{
    int a = 1 ;
    do
    {
        printf ( "Hello World\n" ) ;
        a ++ ;
    } while ( a <= 10 ) ;
    return 0;
}
```

Output

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

WHILE LOOP	DO...WHILE LOOP
It is an entry control loop.	It is an exit control loop.
In while loop expression/condition is checked at the time of entry.	In do... while loop condition is checked at the time of exit.
It does not have semicolon (;) at the end of the expression.	In do while loop semicolon (;) at the end of the expression is compulsory.
Syntax: while(condition) { // loop body }	do { // loop body } while(condition);
Example: while(i<=5) { printf("\n%d",i); i++; }	do { printf("\n%d",i); i++; } while(i<=5) ;

Use of sentinel values

Sentinel Value :

- A sentinel value (also referred to as a flag value, trip value, rogue value, signal value, or dummy data) is a special value in the context of an algorithm which uses its presence as a **condition of termination**, typically in a loop or recursive algorithm.
- **Examples :**
 1. **Null character** for indicating the end of a null-terminated string.
 2. **Null pointer** for indicating the end of a linked list or a tree.
 3. A **negative integer** for indicating the end of a sequence of non-negative integers.

Use of sentinel values

Sentinel Value :

Example :

```
printf("Enter Count ( Or -1 to quit)");  
scanf("%d",count);  
int max=count;  
while(count != -1 )  
{  
    if (count > max)  
        max=count;  
    printf("Enter Count ( Or -1 to quit)");  
    scanf("%d",count);  
}
```

Use of sentinel values

```
if (max > -1)
{
    printf("Maximum is : %d ", max);
}
else
{
    printf("No Count Entered ...");
}
```

Use of sentinel values

- Suppose you want to find the maximum of the data entered from the keyboard.
- It is not known in advanced how many data values a user might want to enter. (And the user may not want to count them!)
- A ***sentinel*** is a special value that is used to detect a special condition, in this case that the user is done entering values.
- The sentinel, of course, must be distinct from any value the user may want to input.

Nesting of looping statements

- A loop inside another loop is called a nested loop.
- The depth of nested loop depends on the complexity of a problem.
- We can have any number of nested loops as required.
- Consider a nested loop where the outer loop runs n times and consists of another loop inside it.
- The inner loop runs m times.
- Then, the total number of times the inner loop runs during the program execution is $n*m$.

Nesting of looping statements

Types of nested loops :

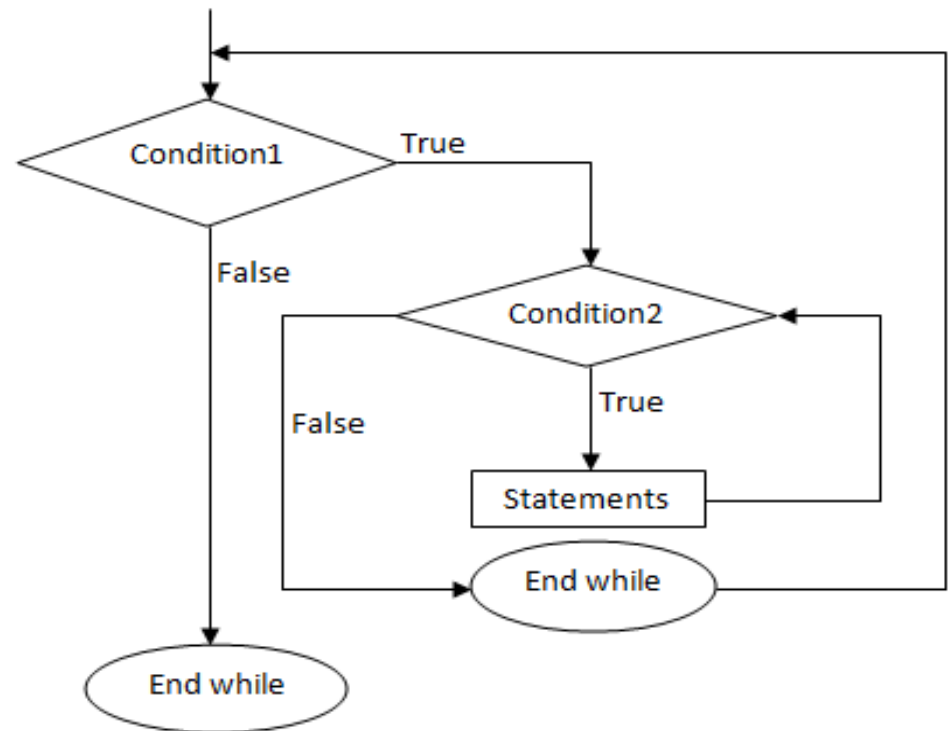
1. Nested while loop
2. Nested do-while loop
3. Nested for loop

Nesting of looping statements

1. Nested while loop :

- A while loop inside another while loop is called nested while loop.
- **Syntax :**

```
while (condition1)
{
    statement(s);
    while (condition2)
    {
        statement(s);
    }
}
```



Nesting of looping statements

1. Nested while loop :

- **Example of Nested while loop :** C program to print the number pattern.

Output :

```
#include <stdio.h>
int main()
{
    int i=1,j;
    while (i <= 5)
    {
        j=1;
        while (j <= i )
        {
            printf("%d ",j);
            j++;
        }
        printf("\n");
        i++;
    }
    return 0;
}
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Nesting of looping statements

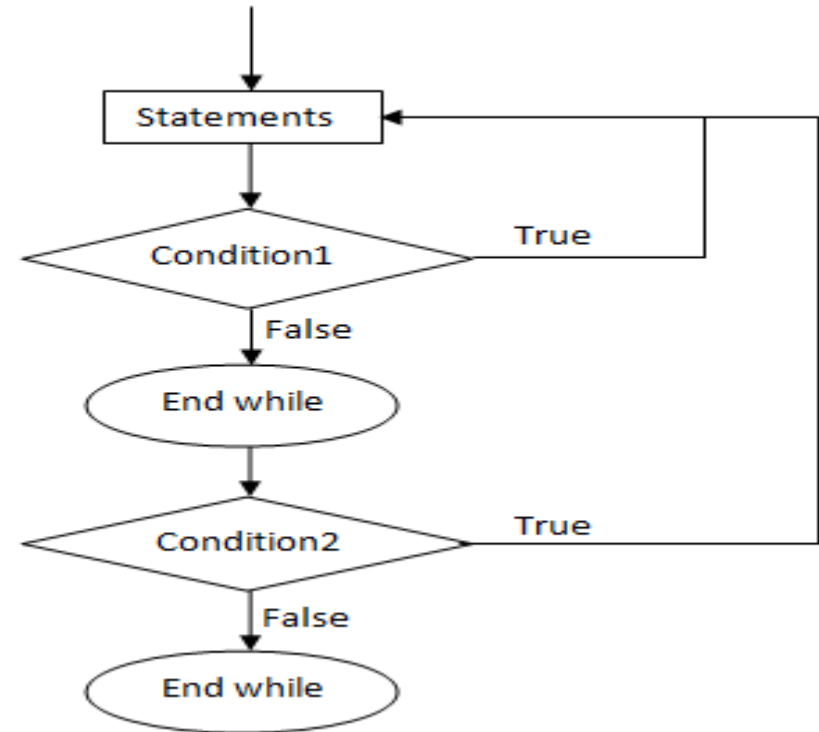
- In this program, nested while loop is used to print the pattern.
- The outermost loop runs 5 times and for every loop, the innermost loop runs i times which is 1 at first, meaning only "1" is printed, then on the next loop it's 2 numbers printing "1 2" and so on till 5 iterations of the loop executes, printing "1 2 3 4 5".
- This way, the given number pattern is printed.

Nesting of looping statements

2. Nested do-while loop :

- A do-while loop inside another do-while loop is called nested do-while loop.
- **Syntax :**

```
do  
{  
    statement(s);  
do {  
    statement(s);  
}while (condition2);  
}while (condition1);
```



Nesting of looping statements

2. Nested do-while loop :

- **Example :** C program to print the given star pattern.

Output :

```
#include <stdio.h>
int main()
{
    int i=1,j;
    do
    {
        j=1;
        do
        {
            printf("*");
            j++;
        }while(j <= i);
        i++;
        printf("\n");
    }while(i <= 5);
    return 0;
}
```

```
*
**
***
****
*****
```

Nesting of looping statements

- In this program, nested do-while loop is used to print the star pattern.
- The outermost loop runs 5 times and for every loop, the innermost loop runs i times which is 1 at first, meaning only one "*" is printed, then on the next loop it's 2 printing two stars and so on till 5 iterations of the loop executes, printing five stars.
- This way, the given star pattern is printed.

Nesting of looping statements

3. Nested for loop :

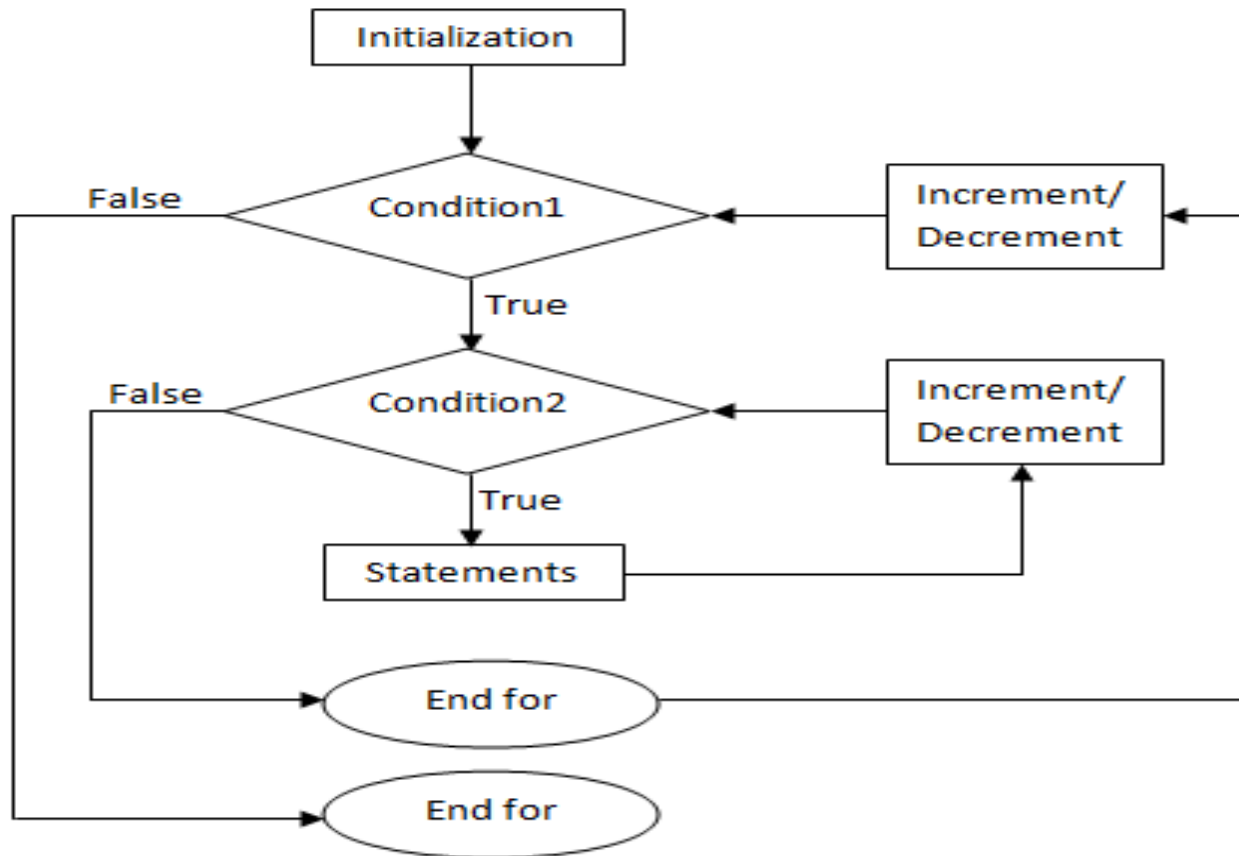
- A for loop inside another for loop is called nested for loop.

- **Syntax :**

```
for (initialization; condition; increment/decrement)
{
    statement(s);
    for (initialization; condition; increment/decrement)
    {
        statement(s);
    }
}
```

Nesting of looping statements

3. Nested for loop :



Nesting of looping statements

3. Nested for loop :

- **Example :** C program to print all the composite numbers from 2 to a certain number entered by user.

```
#include<stdio.h>
#include<math.h>
int main()
{
    int i,j,n;
    printf("Enter a number:");
    scanf("%d",&n);
    for(i=2;i<=n;i++)
    {
        for(j=2;j<=(int)pow(i,0.5);j++)
        {
            if(i%j==0)
            {
                printf("%d is composite\n",i);
                break;
            }
        }
    }
    return 0;
}
```

Output :

```
Enter a number:15
4 is composite
6 is composite
8 is composite
9 is composite
10 is composite
12 is composite
14 is composite
15 is composite
```

Nesting of looping statements

- A number is said to be composite if it has at least one factor other than 1 and itself.
- This program prints all the composite numbers starting from 2 to a certain number n , entered by user.
- We need to use a nested loop to solve this problem.
- The outer for loop runs from 2 to n and the inner loop is used to determine whether a number is composite or not.
- We need to check for that factor starting from 2 to integer part of square root of that number.

Nesting of looping statements

Note

There can be mixed type of nested loop i.e. a for loop inside a while loop, or a while loop inside a do-while loop.

Use of Break & Continue

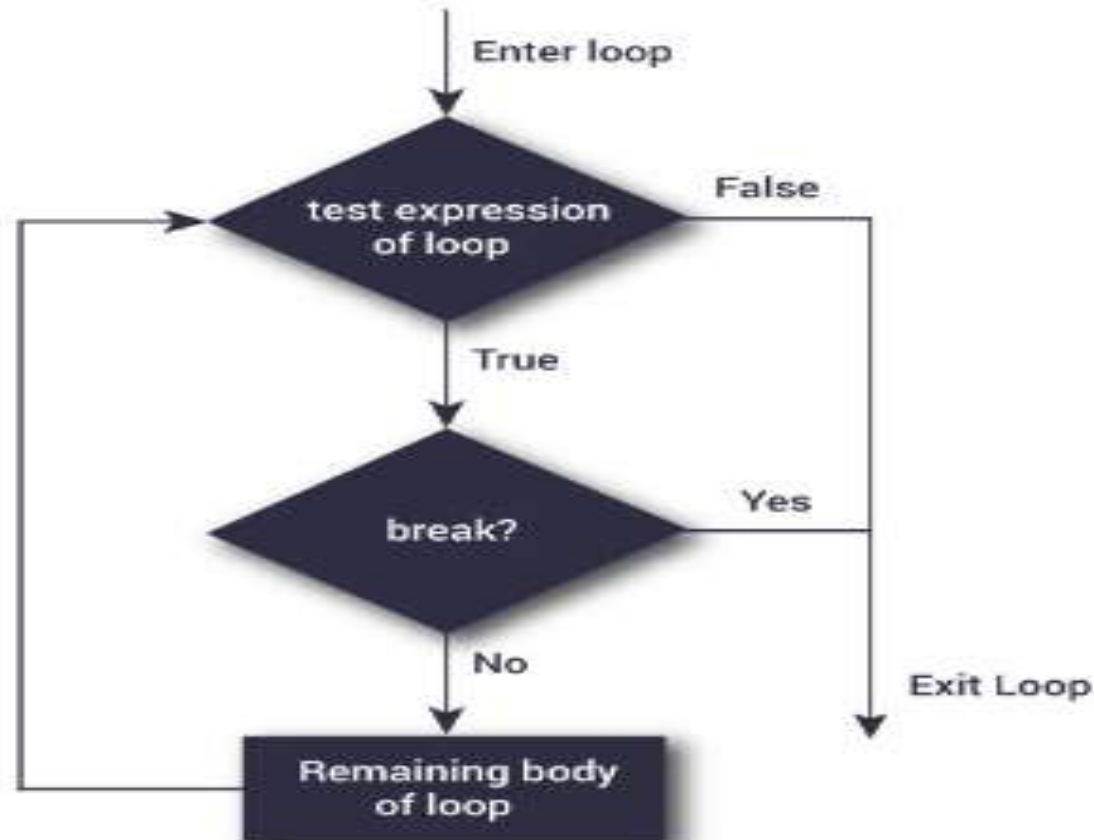
- It is sometimes desirable to **skip some statements** inside the loop or **terminate the loop immediately** without checking the test expression.
- In such cases, break and continue statements are used.

break Statement ::

- The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else.
- **Syntax :**
 break;

Use of Break & Continue

- Flowchart of break statement



Use of Break & Continue

Example :

```
int main()
{
    int i;
    for(i=0; i<10; i++)
    {
        if(i==5)
        {
            printf("\n Coming out of loop when i=5 \n");
            break;
        }
        printf("%d", i);
    }
    return 0;
}
```

Use of Break & Continue

- Output using break; statement :

01234

Coming out of loop when i= 5

- And if we omitted break; statement , the output will be like :

01234

Coming out of loop when i= 5

56789

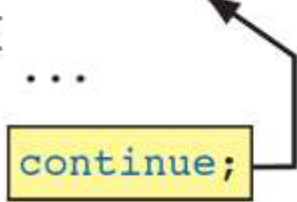
Use of Break & Continue

Continue Statement ::

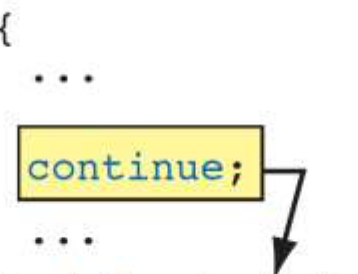
- In C Programs, if we want to take control to the beginning of the loop, by passing the statements inside the loop, which have not yet been executed, in this case we use continue.
- **continue** is reserved keyword in C.
- When **continue** is encountered inside any loop, control automatically passes to the beginning of loop.
- **Syntax :**

continue;

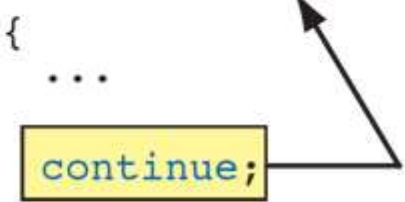
```
while (expr)
{
    ...
    continue;
    ...
} // while
```



```
do
{
    ...
    continue;
    ...
} while (expr);
```

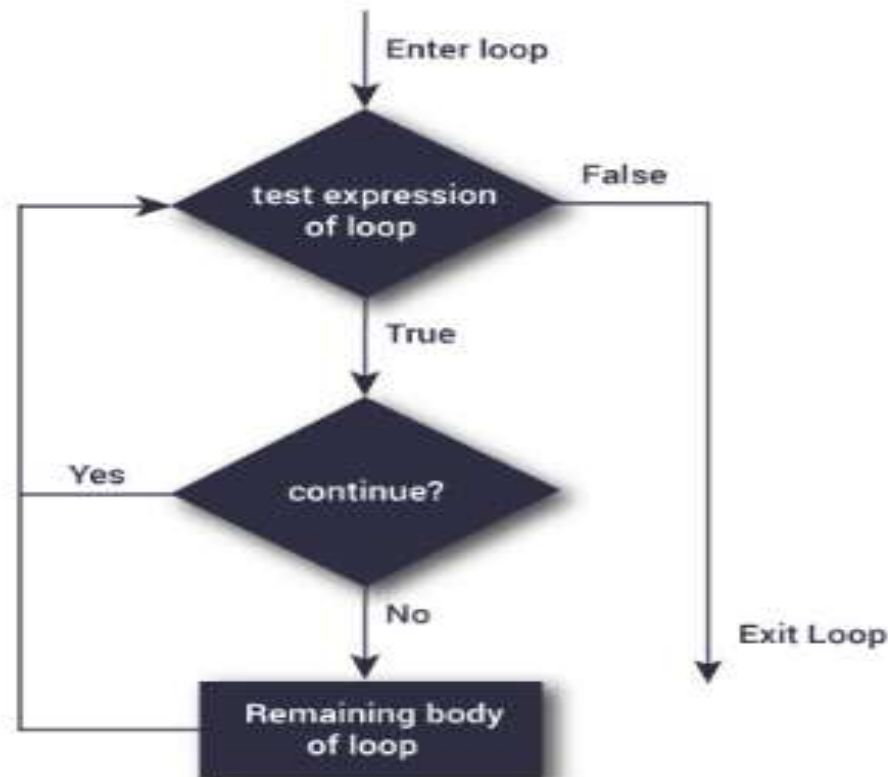


```
for (expr1; expr2; expr3)
{
    ...
    continue;
    ...
} // for
```



Use of Break & Continue

- Flowchart of continue statement



Use of Break & Continue

■ Example :

```
int main()
{
    int i;
    for(i=0; i<10; i++)
    {
        if(i==5)
        {
            printf("\nSkipping %d from display using continue \n", i);
            continue;
        }
        printf("%d", i);
    }
    return 0;
}
```

Use of Break & Continue

- Output using continue; statement :

01234

Skipping 5 from display using continue

6789

- When the value of *i* equals to 5, the continue statement takes the control to if *statement* by passing the value when *i*==5 inside for loop.

Use of if...else in loop

- It is also possible that we could use if..else statement in loop as and when required.
- Example :

```
#include<stdio.h>
```

```
int main()
```

```
{  
    int i = 1; // keep looping  
    while (i < 100)  
    {  
        // if i is even  
        if(i % 2 == 0)  
        {  
            printf("%d ", i);  
        }  
        i++; // increment the number  
    }  
    return 0;  
}
```

Use of if...else in loop

- We have declared a variable `i` and initialized it to 1.
- First, the condition $(i < 100)$ is checked, if it is true.
- Control is transferred inside the body of the while loop. Inside the body of the loop, if condition $(i \% 2 == 0)$ is checked, if it is true then the statement inside the if block is executed.
- Then the value of `i` is incremented using expression `i++`. As there are no more statements left to execute inside the body of the while loop, this completes the first iteration.
- Again the condition $(i < 100)$ is checked, if it is still true then once again the body of the loop is executed.
- This process repeats as long as the value of `i` is less than 100. When `i` reaches 100, the loop terminates and control comes out of the while

Infinite loop

- There may exist some loops that can iterate or occur infinitely.
- These are called **Infinite Loop**.
- These loops occur infinitely because their **condition is always true**.
- We can make an infinite loop by leaving its conditional expression empty. When the conditional expression is empty, it is assumed to be true.
- Example :

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    for(;;)
```

```
    {
```

```
        printf("This is not gonna end!\n");
```

```
    }
```

```
    return 0;
```

```
}
```

Infinite loop

- More Example :

```
int i;  
  
for(i = 0; i < 100; i--)  
{  
    printf("%d\n", i);  
}
```

```
int i = 1  
  
while(i<10)  
{  
    printf("%d\n", i);  
}
```

Infinite loop

Note

To terminate an infinite loop, press Ctrl + C.

Previous year Questions

Fill in the Blanks :

1. In a do ... while loop , if the body of the loop is executed n times , the test condition is evaluated _____ times .
2. The _____ statement is used to skip statement in a loop.
3. A loop that always satisfies the test condition is known as _____ loop.
4. In _____ loop , the entry is automatic and there is only a choice to continue it further or not.
5. When we do not know in advance the number of times the loop will be executed , we use a _____ loop.

Previous year Questions

State True or False :

1. In a while loop , if the body is executed n times , the test condition is executed $(n+1)$ times .
2. It is necessary to have initialization, testing ,and updating expression within the for statement
3. The loop control variable may be updated before or after the loop iterates.
4. The for and do. While loop are the post test loops.
5. When we place a semi colon after the for statement , the compiler will generate an error message .

Previous year Questions

Questions :

1. How is comma operator useful in a for loop? Explain with the help of relevant example?
2. Give the points of similarity and difference between a while loop and do..while loop .
3. Differentiate between the break and the continue statement.
4. In what situations will you prefer to use for , while and do..while loop?